Name : Sujeewa Sandeepa
Topic : Test life cycle
Date : 2021-07-11

Is the process in which the test instance was created, managed and destroyed.

In Junit5 they create a separate test instance for every single method(we write one test per method normally) in a test.

And anyway when it comes to writing tests, we should not depend on previous initializations.

If we want to alter how methods behave, like the order and dependency. We can use JUnit5 hooks. We can use annotations to those kinds of stuff.

Those annotations are,
`@beforeAll`
`@AfterAll`
`@BeforeEach`
`@AfterEach`

Using this to get rid of our past repetitive code,

Before we had to initialize and create a `MathUtils` each and every time when writing a test.

```
@Test
void testAdd() {
      MathUtils mathUtils = new MathUtils();
      int expected = 5;
      int actual = mathUtils.add(3, 2);
      assertEquals(expected, actual, “The add method should add two
numbers”);
}
```

But now with this we can prevent that repetitive code like this,

```
MathUtils mathUtils;
@BeforeEach
void init() {
      mathUtils = new MathUtils();
}
@Test
      void testAdd() {
      int expected = 5;
      int actual = mathUtils.add(3, 2);
      assertEquals(expected, actual, “The add method should add two
numbers”);
}
```

Now this `@BeforeEach` part will run before each and every method, facilitating us to not write it repeatedly, in every single test case.

So when using `@BeforeAll` and @AfterAll it’s going to be a little different and messed up.

As I said before JUnit creates a separate instance, so when using `@BeforAll` it’ll be run even before initializing the class it is in. So the thing is then this will not work because it will not have a class to run in.

So to make this work we need to make one restriction, those methods we run before all and after all have to be ‘static’. If not it won’t work.

```
@BeforeAll
static void beforeAllInit() {
     System.out.print(“This needs to be run before all”);
}
```

Only a static method can be run before an instance is created.